Event Catalog - Kafka Events

Version: 1.0
Date: 2025-11-26
Status: Implementation Ready
Platform: Kafka + Schema Registry (Avro)

---

1. Event-Driven Architecture Overview

```
┌──────────────────────────────────────────────────────────────┐
│          Signature Engine (Producer)                          │
│                                                               │
│  Application Layer                                            │
│       ↓                                                       │
│  OutboxEventPublisher                                         │
│       ↓                                                       │
│  outbox_event table (PostgreSQL)                             │
│       ↓                                                       │
│  Debezium CDC Connector                                      │
└──────────────────────────────────────────────────────────────┘
                    │
                    │
                    ↓
┌──────────────────────────────────────────────────────────────┐
│                    Kafka Cluster                              │
│                                                               │
│  Topics:                                                      │
│   • signature.events (main event stream)                    │
│   • signature.events.dlq (dead letter queue)                │
│                                                               │
│  Schema Registry (Confluent):                                │
│   • Avro schemas with compatibility checks                  │
│   • Versioning: BACKWARD compatibility                       │
└──────────────────────────────────────────────────────────────┘
                    │
                    │
                    ↓
┌──────────────────────────────────────────────────────────────┐
│                    Consumers                                  │
│                                                               │
│  • Analytics Service (stream processing)                     │
│  • Notification Service (email/push to users)               │
│  • Audit Service (long-term storage)                        │
│  • Fraud Detection Service (real-time analysis)             │
│  • Reporting Service (dashboards)                            │
└──────────────────────────────────────────────────────────────┘
```

2. Domain Events Catalog

2.1 Event Types

Event Type	Trigger	Payload	Consumer Use Cases
`SIGNATURE_REQUEST_CREATED`	SignatureRequest created	Request ID, customer, context hash	Analytics, Fraud Detection
`CHALLENGE_SENT`	Challenge delivered to provider	Challenge ID, channel, provider	Monitoring, Notifications
`CHALLENGE_FAILED`	Provider rejected challenge	Challenge ID, error code, channel	Alerting, Fallback trigger
`PROVIDER_FAILED`	Provider unavailable	Provider, error rate, timestamp	Circuit breaker, Degraded mode
`SIGNATURE_COMPLETED`	User completed signature	Request ID, duration, final channel	Analytics, Business metrics
`SIGNATURE_EXPIRED`	TTL reached without completion	Request ID, attempts count	Cleanup, User notification
`SIGNATURE_ABORTED`	User or system aborted	Request ID, reason	Fraud investigation, Support
`ROUTING_RULE_CHANGED`	Admin modified rules	Rule ID, changes	Audit, Config sync

3. Event Schemas (Avro)

3.1 Base Event Schema

Todos los eventos heredan de este schema base:

```
{
  "namespace": "com.bank.signature.events",
```

```json
  "type": "record",
  "name": "BaseEvent",
  "fields": [
    {
      "name": "eventId",
      "type": "string",
      "doc": "Unique event ID (UUIDv7)"
    },
    {
      "name": "aggregateId",
      "type": "string",
      "doc": "SignatureRequest ID"
    },
    {
      "name": "aggregateType",
      "type": "string",
      "doc": "Always 'SignatureRequest'"
    },
    {
      "name": "eventType",
      "type": "string",
      "doc": "Event type enum"
    },
    {
      "name": "occurredAt",
      "type": {
        "type": "long",
        "logicalType": "timestamp-millis"
      },
      "doc": "Event timestamp (epoch millis)"
    },
    {
      "name": "version",
      "type": "string",
      "doc": "Schema version (e.g., '1.0.0')"
    },
    {
      "name": "correlationId",
      "type": ["null", "string"],
      "default": null,
      "doc": "Trace ID for distributed tracing"
    }
  ]
}
```

3.2 SIGNATURE_REQUEST_CREATED

```
{
  "namespace": "com.bank.signature.events",
  "type": "record",
  "name": "SignatureRequestCreated",
  "fields": [
    {
      "name": "eventId",
      "type": "string"
    },
    {
      "name": "aggregateId",
      "type": "string",
      "doc": "SignatureRequest ID"
    },
    {
      "name": "eventType",
      "type": "string",
      "default": "SIGNATURE_REQUEST_CREATED"
    },
    {
      "name": "occurredAt",
      "type": {
        "type": "long",
        "logicalType": "timestamp-millis"
      }
    },
    {
      "name": "version",
      "type": "string",
      "default": "1.0.0"
    },
    {
      "name": "correlationId",
      "type": ["null", "string"],
      "default": null
    },
    {
      "name": "customerId",
      "type": "string",
      "doc": "Pseudonymized customer ID (NO PII)"
    },
    {
      "name": "transactionContextHash",
      "type": "string",
      "doc": "SHA-256 hash of transaction context (for integrity, no PII)"
    },
    {
      "name": "requestedChannel",
      "type": {
        "name": "ChannelType",
        "type": "enum",
        "symbols": ["SMS", "PUSH", "VOICE", "BIOMETRIC"]
      },
      "doc": "Channel determined by routing rules"
    },
    {
      "name": "riskLevel",
      "type": ["null", "string"],
      "default": null,
      "doc": "Risk classification if available (HIGH, MEDIUM, LOW)"
    }
  ]
}
```

3.3 CHALLENGE_SENT

```
{
  "namespace": "com.bank.signature.events",
  "type": "record",
  "name": "ChallengeSent",
  "fields": [
    {
      "name": "eventId",
      "type": "string"
    },
    {
      "name": "aggregateId",
      "type": "string"
    },
    {
      "name": "eventType",
      "type": "string",
      "default": "CHALLENGE_SENT"
    },
    {
      "name": "occurredAt",
      "type": {
        "type": "long",
        "logicalType": "timestamp-millis"
      }
    },
    {
      "name": "version",
      "type": "string",
      "default": "1.0.0"
    },
    {
      "name": "correlationId",
      "type": ["null", "string"],
      "default": null
    },
    {
      "name": "challengeId",
      "type": "string",
      "doc": "Challenge UUID"
    },
    {
      "name": "channelType",
      "type": {
        "name": "ChannelType",
        "type": "enum",
        "symbols": ["SMS", "PUSH", "VOICE", "BIOMETRIC"]
      }
    },
    {
      "name": "provider",
      "type": "string",
      "doc": "Provider name (TWILIO, PUSH_SERVICE, etc.)"
    },
    {
      "name": "providerChallengeId",
      "type": "string",
      "doc": "Provider's unique challenge identifier"
    },
    {
      "name": "attemptNumber",
      "type": "int",
      "doc": "Attempt number (1 = first try, 2+ = fallback)"
    },
    {
      "name": "sentAt",
      "type": {
        "type": "long",
        "logicalType": "timestamp-millis"
      }
    }
  ]
}
```

3.4 CHALLENGE_FAILED

```
{
  "namespace": "com.bank.signature.events",
  "type": "record",
  "name": "ChallengeFailed",
  "fields": [
    {
      "name": "eventId",
      "type": "string"
    },
    {
      "name": "aggregateId",
      "type": "string"
    },
    {
      "name": "eventType",
      "type": "string",
      "default": "CHALLENGE_FAILED"
    },
    {
      "name": "occurredAt",
      "type": {
        "type": "long",
        "logicalType": "timestamp-millis"
      }
    },
    {
      "name": "version",
      "type": "string",
      "default": "1.0.0"
    },
    {
      "name": "correlationId",
      "type": ["null", "string"],
      "default": null
    },
    {
      "name": "challengeId",
      "type": "string"
    },
    {
      "name": "channelType",
      "type": {
        "name": "ChannelType",
        "type": "enum",
        "symbols": ["SMS", "PUSH", "VOICE", "BIOMETRIC"]
      }
    },
    {
      "name": "provider",
      "type": "string"
    },
    {
      "name": "errorCode",
      "type": "string",
      "doc": "Error code from provider or system"
    },
    {
      "name": "errorMessage",
      "type": "string",
      "doc": "Human-readable error message"
    },
    {
      "name": "isRetryable",
      "type": "boolean",
      "doc": "Whether this error allows retry with same channel"
    },
    {
      "name": "willFallback",
      "type": "boolean",
      "doc": "Whether system will attempt fallback to another channel"
    },
    {
      "name": "attemptNumber",
      "type": "int"
    }
  ]
}
```

3.5 PROVIDER_FAILED

```
{
  "namespace": "com.bank.signature.events",
  "type": "record",
  "name": "ProviderFailed",
  "fields": [
    {
      "name": "eventId",
      "type": "string"
    },
    {
      "name": "aggregateId",
      "type": "string",
      "doc": "May be null if system-level failure"
    },
    {
      "name": "eventType",
      "type": "string",
      "default": "PROVIDER_FAILED"
    },
    {
      "name": "occurredAt",
      "type": {
        "type": "long",
        "logicalType": "timestamp-millis"
      }
    },
    {
      "name": "version",
      "type": "string",
      "default": "1.0.0"
    },
    {
      "name": "correlationId",
      "type": ["null", "string"],
      "default": null
    },
    {
      "name": "provider",
      "type": "string"
    },
    {
      "name": "errorRate",
      "type": "double",
      "doc": "Current error rate percentage (0.0 - 100.0)"
    },
    {
      "name": "thresholdExceeded",
      "type": "boolean",
      "doc": "Whether error rate exceeded 50% threshold"
    },
    {
      "name": "degradedMode",
      "type": "boolean",
      "doc": "Whether provider entered degraded mode"
    },
    {
      "name": "degradedUntil",
      "type": ["null", {
        "type": "long",
        "logicalType": "timestamp-millis"
      }],
      "default": null,
      "doc": "Timestamp when provider will be re-enabled (if degraded)"
    }
  ]
}
```

3.6 SIGNATURE_COMPLETED

```
  {
    "namespace": "com.bank.signature.events",
    "type": "record",
    "name": "SignatureCompleted",
    "fields": [
      {
        "name": "eventId",
        "type": "string"
      },
      {
        "name": "aggregateId",
        "type": "string"
      },
      {
        "name": "eventType",
        "type": "string",
        "default": "SIGNATURE_COMPLETED"
      },
      {
        "name": "occurredAt",
        "type": {
          "type": "long",
          "logicalType": "timestamp-millis"
        }
      },
      {
        "name": "version",
        "type": "string",
        "default": "1.0.0"
      },
      {
        "name": "correlationId",
        "type": ["null", "string"],
        "default": null
      },
      {
        "name": "challengeId",
        "type": "string",
        "doc": "Winning challenge ID"
      },
      {
        "name": "finalChannel",
        "type": {
          "name": "ChannelType",
          "type": "enum",
          "symbols": ["SMS", "PUSH", "VOICE", "BIOMETRIC"]
        },
        "doc": "Channel that successfully completed"
      },
      {
        "name": "finalProvider",
        "type": "string"
      },
      {
        "name": "totalAttempts",
        "type": "int",
        "doc": "Total number of challenges sent before success"
      },
      {
        "name": "durationMs",
        "type": "long",
        "doc": "Total duration from creation to completion (milliseconds)"
      },
      {
        "name": "providerProof",
        "type": "string",
        "doc": "Cryptographic proof from provider (for non-repudiation)"
      },
      {
        "name": "completedAt",
        "type": {
          "type": "long",
          "logicalType": "timestamp-millis"
        }
      }
    ]
  }
```

3.7 SIGNATURE_EXPIRED

```
  {
    "namespace": "com.bank.signature.events",
    "type": "record",
    "name": "SignatureExpired",
    "fields": [
      {
        "name": "eventId",
        "type": "string"
      },
      {
        "name": "aggregateId",
        "type": "string"
      },
      {
        "name": "eventType",
        "type": "string",
        "default": "SIGNATURE_EXPIRED"
      },
      {
        "name": "occurredAt",
        "type": {
          "type": "long",
          "logicalType": "timestamp-millis"
        }
      },
      {
        "name": "version",
        "type": "string",
        "default": "1.0.0"
      },
      {
        "name": "correlationId",
        "type": ["null", "string"],
        "default": null
      },
      {
        "name": "totalAttempts",
        "type": "int",
        "doc": "Number of challenges attempted"
      },
      {
        "name": "lastChannel",
        "type": ["null", {
          "name": "ChannelType",
          "type": "enum",
          "symbols": ["SMS", "PUSH", "VOICE", "BIOMETRIC"]
        }],
        "default": null,
        "doc": "Last channel attempted"
      },
      {
        "name": "ttlMinutes",
        "type": "int",
        "doc": "TTL that was configured (default 3 minutes)"
      },
      {
        "name": "expiredAt",
        "type": {
          "type": "long",
          "logicalType": "timestamp-millis"
        }
      }
    ]
  }
```

3.8 SIGNATURE_ABORTED

```
  {
    "namespace": "com.bank.signature.events",
    "type": "record",
    "name": "SignatureAborted",
    "fields": [
      {
        "name": "eventId",
        "type": "string"
      },
      {
        "name": "aggregateId",
        "type": "string"
      },
      {
        "name": "eventType",
        "type": "string",
        "default": "SIGNATURE_ABORTED"
      },
      {
        "name": "occurredAt",
        "type": {
          "type": "long",
          "logicalType": "timestamp-millis"
        }
      },
      {
        "name": "version",
        "type": "string",
        "default": "1.0.0"
      },
      {
        "name": "correlationId",
        "type": ["null", "string"],
        "default": null
      },
      {
        "name": "reason",
        "type": {
          "name": "AbortReason",
          "type": "enum",
          "symbols": ["USER_CANCELLED", "FRAUD_DETECTED", "SYSTEM_ERROR",
"ADMIN_INTERVENTION", "FALLBACK_EXHAUSTED"]
        }
      },
      {
        "name": "reasonDetails",
        "type": ["null", "string"],
        "default": null,
        "doc": "Additional context about abort reason"
      },
      {
        "name": "totalAttempts",
        "type": "int"
      },
      {
        "name": "abortedBy",
        "type": ["null", "string"],
        "default": null,
        "doc": "User or system that aborted"
      }
    ]
  }
```

---

4. Kafka Topic Configuration

4.1 Topic: signature.events

```yaml
topic:
  name: signature.events
  partitions: 12
  replication-factor: 3
  min-insync-replicas: 2
  retention-ms: 604800000  # 7 days
  compression-type: snappy
  cleanup-policy: delete

  config:
    # Ordering guarantee per signature_request_id
    # Partition key = aggregateId (signature_request_id)
    max-message-bytes: 1048576  # 1MB
    segment-ms: 3600000  # 1 hour

    # Durability
    acks: all
    min.insync.replicas: 2
```

4.2 Topic: signature.events.dlq

```yaml
topic:
  name: signature.events.dlq
  partitions: 3
  replication-factor: 3
  retention-ms: 2592000000  # 30 days (longer for investigation)

  # Dead letter queue for:
  # - Deserialization errors
  # - Consumer processing failures (after retries)
  # - Schema validation failures
```

4.3 Partitioning Strategy

```java
// Partitioning por aggregateId para garantizar orden
public class SignatureEventPartitioner extends DefaultPartitioner {
    @Override
    public int partition(String topic, Object key, byte[] keyBytes,
                         Object value, byte[] valueBytes, Cluster cluster) {
        // Key = aggregateId (signature_request_id)
        // Todos los eventos de un SignatureRequest van a la misma partición
        return Math.abs(key.hashCode()) % cluster.partitionCountForTopic(topic);
    }
}
```

Garantía: Todos los eventos de un `SignatureRequest` se procesan en orden.

---

5. Debezium Configuration

5.1 Outbox Connector

```json
{
  "name": "signature-outbox-connector",
  "config": {
    "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
    "tasks.max": "1",
    "database.hostname": "${DB_HOST}",
    "database.port": "5432",
    "database.user": "${DB_USER}",
    "database.password": "${DB_PASS}",
    "database.dbname": "signature_db",
    "database.server.name": "signature-server",
    "plugin.name": "pgoutput",

    "table.include.list": "public.outbox_event",
    "publication.name": "signature_outbox_publication",
    "publication.autocreate.mode": "filtered",

    "transforms": "outbox",
    "transforms.outbox.type": "io.debezium.transforms.outbox.EventRouter",
    "transforms.outbox.table.field.event.id": "id",
    "transforms.outbox.table.field.event.key": "aggregate_id",
    "transforms.outbox.table.field.event.type": "event_type",
    "transforms.outbox.table.field.event.payload": "payload",
    "transforms.outbox.table.field.event.timestamp": "created_at",
    "transforms.outbox.route.topic.replacement": "signature.events",

    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "io.confluent.connect.avro.AvroConverter",
    "value.converter.schema.registry.url": "${SCHEMA_REGISTRY_URL}",

    "tombstones.on.delete": "false",
    "snapshot.mode": "never",
    "publication.autocreate.mode": "filtered"
  }
}
```

5.2 PostgreSQL Publication

```sql
-- Crear publication para Debezium
CREATE PUBLICATION signature_outbox_publication
    FOR TABLE outbox_event;

-- Verificar replication slot
SELECT * FROM pg_replication_slots;
```

6. Schema Evolution Strategy

6.1 Compatibility Rules

- **Schema Registry**: Confluent Schema Registry
- **Compatibility Mode**: `BACKWARD` (consumers nuevos pueden leer eventos viejos)
- **Versioning**: Semantic versioning en campo `version`

6.2 Adding Fields (Safe)

```
{
  "name": "newField",
  "type": ["null", "string"],
  "default": null,
  "doc": "New optional field"
}
```

✅ **Backward compatible**: Consumers viejos ignoran campo nuevo.

6.3 Removing Fields (Breaking)

❌ **Breaking change**: Requiere migración de consumers primero.

Estrategia:

1. Deprecar campo en schema docs
2. Deployar consumers que no usen el campo
3. Nueva versión de schema sin el campo
4. Deployar producers actualizados

6.4 Schema Validation

```
@Service
public class EventPublisher {

    private final SchemaRegistryClient schemaRegistry;
    private final KafkaTemplate<String, GenericRecord> kafkaTemplate;

    public void publish(DomainEvent event) {
        // Validar schema antes de publicar
        Schema schema = schemaRegistry.getLatestSchemaMetadata("signature.events-
value")
            .getSchema();

        GenericRecord avroRecord = eventToAvro(event, schema);
        // Kafka producer validará contra schema registry
        kafkaTemplate.send("signature.events", event.getAggregateId(),
avroRecord);
    }
}
```

---

7. Event Ordering Guarantees

7.1 Per-Aggregate Ordering

```
Partition Key = aggregateId (signature_request_id)

SignatureRequest A:
  ├─ SIGNATURE_REQUEST_CREATED  → Partition 3
  ├─ CHALLENGE_SENT             → Partition 3
  ├─ CHALLENGE_FAILED           → Partition 3
  └─ SIGNATURE_COMPLETED        → Partition 3

SignatureRequest B:
  ├─ SIGNATURE_REQUEST_CREATED  → Partition 7
  └─ SIGNATURE_EXPIRED          → Partition 7
```

✅ **Garantía**: Eventos del mismo `SignatureRequest` procesados en orden.

7.2 Global Ordering

❌ **No garantizado**: Eventos de diferentes `SignatureRequest` pueden procesarse en cualquier orden.

Razón: Performance y escalabilidad (12 particiones).

---

8. Consumer Groups

8.1 Analytics Consumer

```yaml
consumer:
  group-id: signature-analytics-group
  auto-offset-reset: earliest
  enable-auto-commit: false  # Manual commit after processing
  max-poll-records: 500

  topics:
    - signature.events

  processing:
    - Stream to data warehouse (Snowflake/BigQuery)
    - Real-time dashboards (Grafana)
    - Cost optimization calculations
```

8.2 Notification Consumer

```
consumer:
  group-id: signature-notification-group
  auto-offset-reset: latest  # Solo eventos nuevos
  enable-auto-commit: true

  topics:
    - signature.events

  filters:
    - SIGNATURE_COMPLETED → Email confirmation
    - SIGNATURE_EXPIRED → Retry notification
    - CHALLENGE_FAILED → Support alert
```

8.3 Audit Consumer

```
consumer:
  group-id: signature-audit-group
  auto-offset-reset: earliest
  enable-auto-commit: false

  topics:
    - signature.events

  processing:
    - Store in immutable audit log (S3/GCS)
    - Compliance reporting
    - Legal non-repudiation evidence
```

---

9. Monitoring & Observability

9.1 Kafka Metrics

```
metrics:
  producers:
    - signature-engine.record-send-rate
    - signature-engine.record-error-rate
    - signature-engine.request-latency-avg

  consumers:
    - analytics-consumer.records-lag
```

```yaml
    - analytics-consumer.records-consumed-rate
    - notification-consumer.commit-latency-avg

  topics:
    - signature.events.bytes-in-per-sec
    - signature.events.messages-in-per-sec
    - signature.events.under-replicated-partitions
```

9.2 Alerts

```yaml
alerts:
  - name: HighConsumerLag
    condition: records-lag > 10000
    severity: critical
    action: page-oncall

  - name: OutboxEventsNotPublished
    condition: outbox_event WHERE published_at IS NULL > 100
    severity: high
    action: alert-engineering

  - name: DeadLetterQueueGrowth
    condition: signature.events.dlq message-count > 50
    severity: medium
    action: investigate
```

---

10. Security

10.1 Encryption

```yaml
kafka:
  security:
    protocol: SASL_SSL
    sasl-mechanism: SCRAM-SHA-512
    ssl:
      truststore-location: /etc/kafka/truststore.jks
      truststore-password: ${TRUSTSTORE_PASSWORD}

  # Encryption in transit (TLS)
  ssl-enabled: true

  # Encryption at rest (broker-side)
  log-encryption: true
```

10.2 ACLs (Access Control Lists)

```
# Producer (Signature Engine)
kafka-acls --add --allow-principal User:signature-engine \
  --operation Write \
  --topic signature.events \
  --cluster

# Consumer (Analytics)
kafka-acls --add --allow-principal User:analytics-service \
  --operation Read \
  --topic signature.events \
  --group signature-analytics-group \
  --cluster
```

10.3 Data Masking

```
// NEVER include PII in events
public class EventSanitizer {

    public static String hashTransactionContext(TransactionContext context) {
        // SHA-256 hash for integrity, no PII
        return DigestUtils.sha256Hex(context.toJson());
    }

    public static String pseudonymizeCustomerId(String realCustomerId) {
        // Already pseudonymized in DB, double-check
        if (containsPII(realCustomerId)) {
            throw new SecurityException("PII detected in event payload");
        }
        return realCustomerId;
    }
}
```

11. Testing Events

11.1 Event Publishing Test

```
@SpringBootTest
@Testcontainers
class EventPublishingIT {

    @Container
    static KafkaContainer kafka = new KafkaContainer(
        DockerImageName.parse("confluentinc/cp-kafka:7.5.0")
    );
    @Test
    void shouldPublishSignatureRequestCreatedEvent() {
        // Given
        SignatureRequest request = createTestRequest();

        // When
        signatureService.create(request);

        // Then - Verify outbox
        OutboxEvent outboxEvent =
outboxRepository.findByAggregateId(request.getId());

 assertThat(outboxEvent.getEventType()).isEqualTo("SIGNATURE_REQUEST_CREATED");

        // And - Verify Kafka (after Debezium processes)
        ConsumerRecord<String, GenericRecord> record =
            consumeEvent("signature.events", Duration.ofSeconds(10));

        assertThat(record.key()).isEqualTo(request.getId().toString());

 assertThat(record.value().get("eventType")).isEqualTo("SIGNATURE_REQUEST_CREATED"
);
    }
}
```

11.2 Schema Validation Test

```
class EventSchemaTest {

    @Test
    void eventsShouldConformToAvroSchema() throws IOException {
        Schema schema = new Schema.Parser().parse(
            new File("src/main/resources/kafka/schemas/signature-event.avsc")
        );

        SignatureRequestCreated event = createTestEvent();
        GenericRecord avroRecord = eventToAvro(event);

        // Validate against schema
        assertDoesNotThrow(() -> {
            GenericDatumWriter<GenericRecord> writer =
                new GenericDatumWriter<>(schema);
            writer.write(avroRecord, new NullEncoder());
        });
    }
}
```

Status: ✅ **COMPLETE - READY FOR KAFKA SETUP**

Next Steps:

- Deploy Kafka cluster (Confluent Cloud o self-hosted)
- Configure Schema Registry
- Deploy Debezium connector
- Implement event consumers